Hypertree Decomposition for Parallel Constraint Solving

Ke Liu, Sven Löffler, Petra Hofstedt

liuke@b-tu.de
Brandenburg University of Technology Cottbus-Senftenberg

KPS 2017
Outline

1 Hypertree decomposition
   • Background
   • Foundations of Det-k-CP

2 More details of Det-k-CP
   • The Algorithms of Det-k-CP
   • Experimental Results

3 Summary
Outline

1. Hypertree decomposition
   - Background
     - Foundations of Det-k-CP

2. More details of Det-k-CP
   - The Algorithms of Det-k-CP
     - Experimental Results

3. Summary
Introduction

This presentation tries to briefly answer the following questions:

1. How do we solve a given constraint network in parallel?

- Parallel search
- Parallel consistency
- Parallel constraint solving

2. Why do we need to do decomposition?
- If we divide the original intractable problem into the tractable sub-problems because many NP-complete and NP-hard problems can be solved in polynomial time if the corresponding hypergraph has the bounded hypertree-width [3].
- The entire constraint network is dead-end free if it can achieve directional arc-consistency [1].
Introduction

This presentation tries to briefly answer the following questions:

1. How do we solve a given constraint network in parallel?
   - Parallel search
   - Parallel consistency
   - Parallel constraint solving

2. Why do we need to do decomposition?
Introduction

This presentation tries to briefly answer the following questions:

1. How do we solve a given constraint network in parallel?
   - Parallel search
   - Parallel consistency
   - Parallel constraint solving

2. Why do we need to do decomposition?
   - If we divide the original intractable problem into the tractable sub-problems because many NP-complete and NP-hard problems can be solved in polynomial time if the corresponding hypergraph has the bounded hypertree-width [3].
   - The entire constraint network is dead-end free if it can achieve directional arc-consistency [1].
Introduction

This presentation tries to briefly answer the following questions:

1. How do we solve a given constraint network in parallel?
   - Parallel search
   - Parallel consistency
   - **Parallel constraint solving**

2. Why do we need to do decomposition?
   - If we divide the original intractable problem into the tractable sub-problems because many NP-complete and NP-hard problems can be solved in polynomial time if the corresponding hypergraph has the bounded hypertree-width [3].
   - The entire constraint network is dead-end free if it can achieve directional arc-consistency [1].
Introduction

3 What is the motivation of the new decomposition algorithm?

- The target decomposition tree of existing algorithms are the hypertree with width as small as possible because the small hypertree width indicates the problem can be solved faster. The following situation could be happened!
What is the motivation of the new decomposition algorithm?

- The target decomposition tree of existing algorithms are the hypertree with width as small as possible because the small hypertree width indicates the problem can be solved faster. The following situation could be happened!

![Decomposition Trees](image_url)

decomposed by det-k-decomp [3] 

decomposed by det-k-CP
How does Det-k-CP work?

- We will answer this question in the following slides.
Structural Decomposition Methods

The existing decomposition methods for constraint network

- Join-Tree-Clustering
- Cycle-cutset Decomposition
- Hinge Decomposition
- Hypertree Decomposition

The main objective of Hypertree decomposition is to obtain a decomposition graph as small hypertree width as possible.
Hypertree decomposition example

- \textit{allDifferent1}(x_3, x_4, x_5, x_7)
- \textit{allDifferent2}(x_1, x_4, x_6, x_9)
- \textit{atLeastNvalues}(x_1, x_2, x_3)
- \textit{arithm2}(x_6, x_8)
- \textit{table1}(x_5, x_8, x_{10})
- \textit{table2}(x_7, x_8, x_9)
- \textit{arithm1}(x_5, x_6)
The hypergraph for the constraint network.
The example is based on [2]
Hypertree decomposition

\[
\{ \text{allDifferent1, allDifferent2} \}, \{ x_1, x_4, x_6, x_9, x_3, x_5, x_7 \}
\]

\[
\{ \text{atLeastNValues} \}, \{ x_1, x_2, x_3 \}
\]

\[
\{ \text{table2, arithm1} \}, \{ x_5, x_6, x_7, x_8, x_9 \}
\]

\[
\{ \text{arithm2, table1} \}, \{ x_5, x_6, x_8, x_10 \}
\]

One possible hypertree decomposition for the hypergraph on the previous page
Outline

1 Hypertree decomposition
   • Background
   • Foundations of Det-k-CP

2 More details of Det-k-CP
   • The Algorithms of Det-k-CP
   • Experimental Results

3 Summary
The design objective of \textit{det-k-CP} is a mapping algorithm for parallel constraint solving.

- A reasonable execution time

\[ c_1, ..., c_i, ..., c_j, ..., c_k, ..., c_l, ..., c_m, ..., c_n, ..., c_o \]

A degenerate decomposition tree for parallel solving.
The Objective of det-k-CP

- The design objective of *det-k-CP* is a mapping algorithm for parallel constraint solving.
  - A reasonable execution time
  - Relatively even distribution for multi-core processor.
The Objective of det-k-CP

- The design objective of *det-k-CP* is a mapping algorithm for parallel constraint solving.
  - A reasonable execution time
  - Relatively even distribution for multi-core processor.
  - Accords with hypertree structure
The Objective of det-k-CP

- The design objective of det-k-CP is a mapping algorithm for parallel constraint solving.
  - A reasonable execution time
  - Relatively even distribution for multi-core processor.
  - **Accords with hypertree structure**
The Objective of det-k-CP

- The design objective of *det-k-CP* is a mapping algorithm for parallel constraint solving.
  - A reasonable execution time
  - Relatively even distribution for multi-core processor.
  - **Accords with hypertree structure**

A degenerate decomposition tree for parallel solving.
An acyclic constraint network can be solved efficiently. [1]

- Because the constraint problem has a join-tree, its dual problem is a tree of binary constraints and can therefore be solved by the tree-solving algorithm [1].
An Important Property

One edge between two nodes can be eliminated without changing the set of all solutions for the constraint network because there exists an alternative path between the two nodes.
An Important Property

One edge between two nodes can be eliminated without changing the set of all solutions for the constraint network because there exists an alternative path between the two nodes.
The Principle of det-k-CP

\[
\begin{align*}
&c_1(x_1, x_2, x_3) \\
&c_2(x_3, x_4, x_5) \\
&c_3(x_2, x_5, x_6)
\end{align*}
\]

The original simple constraint network
The Principle of det-k-CP

The original simple constraint network

Adding procedure for eliminating non-adjacent edge
The problem caused by simply adding constraint

The arc that had been removed before might occur again due to the subsequent adding procedure.
Outline

1 Hypertree decomposition
   - Background
   - Foundations of Det-k-CP

2 More details of Det-k-CP
   - The Algorithms of Det-k-CP
   - Experimental Results

3 Summary
Two conditions of Det-k-CP

Mathematically, a decomposed graph after decomposition by \textit{det-k-CP} must meet the following two conditions:

- The shared variables between any pair of nodes \((N_p, N_{p+1})\) in the interval \([i..j]\) must contain the shared variables between \((N_i, N_j)\), where \(i \leq p < p + 1 \leq j\).
Two conditions of Det-k-CP

Mathematically, a decomposed graph after decomposition by *det-k-CP* must meet the following two conditions:

1. The shared variables between any pair of nodes \((N_p, N_{p+1})\) in the interval \([i..j]\) must contain the shared variables between \((N_i, N_j)\), where \(i \leq p < p + 1 \leq j\).

2. The decomposition method *det-k-CP* does not lose any constraint or variable.
The main procedure of Det-k-CP

Algorithm 1: The main procedure of Det-k-CP

Input: A Constraint Network $N$, and the desired number of nodes $k$.

1. Set a list which contains sorted constraints of $N$ based on weight.
2. Evenly distribute constraint onto $k$ nodes.
3. while true do
   4. get a Potential Solution;
   5. if The graph passes test conditions (1) and (2) then
      6. break;
   7. end
   8. Swap nodes in array.Nodes;
3. end
10. return a degenerated decomposition tree;
Get a potential solution for Det-k-CP

**Algorithm 2: getPotentialSolution**

**Output:** A potential solution

1. Set $i_{len}$ = the number of nodes;
2. Set $i_{start} = i_{len}$-2;
3. **while** $i_{start} \geq 0$ **do**
   4. **for** $i_{end} \leftarrow i_{start} + 2$ **to** $i_{len}$ **do**
   5. \hspace{1em} eliminateEdge($i_{start}$, $i_{end}$, array_Nodes) ;
   6. **end**
   7. Set $i_{start} = i_{start} - 1$ ;
4. **end**
Eliminate an Edge in Det-k-CP

**Algorithm 3:** eliminate an Edge between node

**Input:** $i_{\text{start}}$, $i_{\text{end}}$, array\_Nodes

1. for $i \leftarrow i_{\text{start}}$ to $i_{\text{end}} - 1$ do
   2. Set $\text{list\_2BeEliminated} = \text{getSharedVariables}(i_{\text{start}}, i_{\text{end}}, \text{array\_Nodes})$;
   3. Set $j = i + 1$;
   4. Set $\text{list\_SharedOnMainPath} = \text{getSharedVariables}(i, j, \text{array\_Nodes})$;
   5. if $\text{list\_SharedOnMainPath}.\text{notContainsAll}(\text{list\_2beEliminate})$ then
      6. $\text{list\_2BeEliminated}.\text{removeAll}(\text{list\_SharedOnMainPath})$;
      7. Set $\text{list\_2beAddedConstraints} = \text{getMinimumSetConstraints4Shared}(i, \text{array\_Nodes},$
         $\text{list\_2BeEliminated})$;
      8. foreach constraint $cs \in \text{list\_2beAddedConstraints}$ do
         9. if $\text{array\_Nodes}[j]\text{notContained} cs$ then
            10. add $cs$ into $\text{array\_Nodes}[j]$;
         end
      end
   end
end
Time complexity of Det-k-CP

The asymptotic time of Det-k-CP is $O(k^4 \cdot N_c \cdot (N_c + N_v))$, where $k$ is the number of nodes in the desired degenerated decomposition tree, $N_c$ and $N_v$ are the number of constraints and the number of variables for the given constraint network respectively.
Outline

1 Hypertree decomposition
   • Background
   • Foundations of Det-k-CP

2 More details of Det-k-CP
   • The Algorithms of Det-k-CP
     • Experimental Results

3 Summary
The benchmark suit for testing det-k-CP is extracted from practical industrial constraint problem [3].

The algorithms are implemented in Java.

All the experiments are set up on an iMac computer having an Intel i7-3770 CPU, 3.40GHz, with 8 GB 1600 MHz DDR3.
Results and Conclusion of the experiment

- For the application of parallel constraint solving, the algorithm can be applied to medium scale constraints network with the number of constraints ranging from around 400 to 700.
Results and Conclusion of the experiment

- For the application of parallel constraint solving, the algorithm can be applied to medium scale constraints network with the number of constraints ranging from around 400 to 700.
- Competitive execution time as long as the hypergraphs are sufficiently simple. e.g., the constraint network $s5378$ with 2958 constraints and 2993 variables) and a big $k$ (e.g., $k = 16$) takes around two minutes.
Conclusion and Future Work

- The possible ways for improving det-k-CP
  - Take into consideration an estimate of the amount/complexity of computation for the constraints
  - Local search methods might be employed to replace the existing stochastic strategies in order to obtain more optimized decomposition result.
  - Add constraints to one node from other nodes covering the same variables.

Future Work

- To develop a related join selection algorithm for det-k-CP to gain the speed up.
Conclusion and Future Work

- The possible ways for improving det-k-CP
  - Take into consideration an estimate of the amount/complexity of computation for the constraints
  - Local search methods might be employed to replace the existing stochastic strategies in order to obtain more optimized decomposition result.
  - Add constraints to one node from other nodes covering the same variables.

- Future Work
  - To develop a related join selection algorithm for det-k-CP to gain the speed up.
Thank you for your attention!
Question?
References

